Improving the Encoding of LTL Model Checking into SAT*

Alessandro Cimatti[1], Marco Pistore[1], Marco Roveri[1], and Roberto Sebastiani[1,2]

[1] ITC-IRST, Trento, Italy
{cimatti,pistore,roveri}@irst.itc.it
[2] Dept. of Information and Communication Technology – University of Trento, Italy
rseba@science.unitn.it

Abstract. Bounded Model Checking (BMC) is a technique for encoding an LTL model checking problem into a problem of propositional satisfiability. Since the seminal paper by Biere et al. [2], the research on BMC has been primarily directed at achieving higher efficiency for solving reachability properties. In this paper, we tackle the problem of improving BMC encodings for the full class of LTL properties. We start noticing some properties of the encoding of [2], and we exploit them to define improvements that make the resulting boolean formulas smaller or simpler to solve.

1 Motivations and goals

Model Checking [8, 7] is a powerful technique for verifying systems and detecting errors at early stages of the design process, which is obtaining wide acceptance in industrial settings. In Model Checking, the specification is expressed in temporal logic —either Computation Tree Logic (CTL) or Linear temporal Logic (LTL)— and the system is modeled as a finite state machine (FSM). A traversal algorithm verifies exhaustively whether the FSM satisfies the property or not. Symbolic Model Checking uses Ordered Boolean Decision Diagrams (BDDs) [4] to encode the FSM [5, 11].

Recently a new approach for Symbolic Model Checking has been proposed, called Bounded Model Checking (BMC), which is based on SAT techniques [2]. Given a FSM M and an LTL specification f, the idea is to look for counter-examples of maximum length k, and to generate a boolean formula which is satisfiable if and only if such counter-example exists. The boolean formula is then given as input to a SAT solver. If the formula is satisfiable, the satisfying assignment returned is converted into a counter-example execution path.

* The fourth author is sponsored under the MURST COFIN99 project "Model checking and satisfiability: development of novel decision procedures and comparative evaluation and experimental analysis in significant application areas – Moses", protocol number 9909261583. The first, second and fourth authors are sponsored by the CALCULEMUS! IHP-RTN EC project, contract code HPRN-CT-2000-00102, and have thus benefited of the financial contribution of the Commission through the IHP program.

Since then, the research on bounded model checking has mainly focused on using effective data structures to encode boolean expression [1, 16] or on customizing SAT procedures for BMC problems [14]. Moreover, most work have restricted to the very particular subproblem of reachability [3, 1, 14, 9]. For the general case, no alternative encoding than that in [2] has been proposed so far.

In this paper, we analyze the encoding of [2], we reveal and prove some basic properties of the encoding, and we use these properties to define some improvements which make the resulting boolean formula smaller or simpler to solve. Some of these improvements are currently implemented in the NuSMV symbolic model checker [6] (NuSMV is available at `http://nusmv.irst.itc.it`).

2 The basic encoding

We briefly recall some basic notions and a description of the BMC encoding, as proposed in [2]. We omit any formal description of the semantics of LTL and of LTL model checking, which can be found there.

We consider LTL formulas in negative normal form, which are defined as follows: a propositional literal is a LTL formula; if h and g are LTL formulas, then $h \wedge g$, $h \vee g$, $\mathbf{X}g$, $\mathbf{G}g$, $\mathbf{F}g$, $h\mathbf{U}g$ and $h\mathbf{R}g$ are LTL formulas, $\mathbf{X}$, $\mathbf{G}$, $\mathbf{F}$, $\mathbf{U}$ and $\mathbf{R}$ being the standard "next", "globally", "eventually", "until" and "releases" temporal operators respectively. We denote by $depth(f)$ the maximum level of nesting of temporal operators in f. A Kripke Structure M is a tuple $\langle S, I, T, \mathcal{L} \rangle$ with a finite set of states S, a set of initial states $I \subset S$, a transition relation $T \subseteq S \times S$ and a labeling function $\mathcal{L} : S \to \mathcal{P}(\mathcal{A})$, $\mathcal{A}$ being the set of atomic propositions.

Given M, an LTL formula f and an integer $k \geq 0$, the existential bounded model checking problem $M \models_k \mathbf{E}f$, meaning "there exist an execution path of M of length k satisfying the temporal property f", is equivalent to the satisfiability problem of a boolean formula $[[M, f]]_k$ defined as follows:

$$[[M, f]]_k := [[M]]_k \wedge [[f]]_k \tag{1}$$

where

$$[[M]]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}), \tag{2}$$

$$[[f]]_k := (\neg L_k \wedge [[f]]_k^0) \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[[f]]_k^0), \tag{3}$$

${}_lL_k := T(s_k, s_l)$, $L_k := \bigvee_{l=0}^{k} {}_lL_k$, $[[f]]_k^i$ and ${}_l[[f]]_k^i$ are described in Table 1. (If f is boolean, we denote by f_i the value of f at step i.) Intuitively,

- $[[M, f]]_k$ represents the paths of length k which are compatible with the initial conditions and the transition relation, and satisfy f;

f	$[[f]]_k^i$	${}_l[[f]]_k^i$
p	p_i	p_i
$\neg p$	$\neg p_i$	$\neg p_i$
$h \wedge g$	$[[h]]_k^i \wedge [[g]]_k^i$	${}_l[[h]]_k^i \wedge {}_l[[g]]_k^i$
$h \vee g$	$[[h]]_k^i \vee [[g]]_k^i$	${}_l[[h]]_k^i \vee {}_l[[g]]_k^i$
$\mathbf{X}g$	$[[g]]_k^{i+1}$ $if\ i < k$ $\perp$ $otherwise.$	${}_l[[g]]_k^{i+1}$ $if\ i < k$ ${}_l[[g]]_k^{l}$ $otherwise.$
$\mathbf{G}g$	$\perp$	$\bigwedge_{j=min(i,l)}^{k} {}_l[[g]]_k^j$
$\mathbf{F}g$	$\bigvee_{j=i}^{k} [[g]]_k^j$	$\bigvee_{j=min(i,l)}^{k} {}_l[[g]]_k^j$
$h\mathbf{U}g$	$\bigvee_{j=i}^{k} \left([[g]]_k^j \wedge \bigwedge_{n=i}^{j-1} [[h]]_k^n \right)$	$\bigvee_{j=i}^{k} \left({}_l[[g]]_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l[[h]]_k^n \right) \vee$ $\bigvee_{j=l}^{i-1} \left({}_l[[g]]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[[h]]_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l[[h]]_k^n \right)$
$h\mathbf{R}g$	$\bigvee_{j=i}^{k} \left([[h]]_k^j \wedge \bigwedge_{n=i}^{j} [[g]]_k^n \right)$	$\bigwedge_{j=min(i,l)}^{k} {}_l[[g]]_k^j \vee$ $\bigvee_{j=i}^{k} \left({}_l[[h]]_k^j \wedge \bigwedge_{n=i}^{j} {}_l[[g]]_k^n \right) \vee$ $\bigvee_{j=l}^{i-1} \left({}_l[[h]]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[[g]]_k^n \wedge \bigwedge_{n=l}^{j} {}_l[[g]]_k^n \right)$

Table 1. Recursive definition of $[[f]]_k^i$ and ${}_l[[f]]_k^i$.

- $[[M]]_k$ represents the paths which are compatible with the initial conditions and the transition relation;
- $[[f]]_k$ represents the paths which satisfy f;
- ${}_lL_k$ represents the transition from step k to step l, which induces a loop;
- L_k represents the disjunction of every transitions from step k to a step l;
- $[[f]]_k^0$ represents the paths which satisfy f, if there is no loop from step k to any step $l \leq k$;
- ${}_l[[f]]_k^0$ represents the paths which satisfy f, if there is a loop from step k to step l.

Of course, the method is not complete, in the sense that, if $[[M,f]]_k$ is unsatisfiable (that is, $M \not\models_k \mathbf{E}f$) then nothing can be said about the existence of paths of length $\geq k$. [1] Of course, there is a maximum value of k, called the *diameter* of the problem, after which we can conclude there is no solution. Unfortunately, such value is typically very big and very hard to compute [3]. Thus, the typical technique is to generate and solve $[[M,f]]_k$ for increasing values of $k = 0, 1, 2, 3, \ldots$, until either a satisfying path is found, or a given timeout is reached.

Notice that, a path of length k satisfying an LTL formula f (i.e. such that $M \models_k \mathbf{E}f$) corresponds to a counter-example for the universal model checking problem $M \models \mathbf{A}\neg f$, meaning "for all computation path the LTL property $\neg f$ is satisfied".

[1] This should not be a surprise, as LTL model checking is PSPACE-complete, while SAT is NP-complete.

3 Problems with the basic encoding

In this paper we assume that all the boolean formulas are represented by binary directed acyclic graphs (DAGs) so that all common subformulas are shared. [2] These formulas keep reasonably small, but are not canonical, in the sense that the DAG representation of logically equivalent formulas is not unique. The DAG representing $[[M, f]]_k$ is CNF-ized by means of a labeling CNF conversion (see. e.g., [12, 10]). Remarkably, this conversion avoids the exponential explosion in size, but forces the introduction of new boolean variables. Moreover, the resulting formula is not logically equivalent to the previous one, but it is only equally satisfiable.

A key problem with the encoding described in Section 2 is that in many cases it produces redundant formulas. For instance, if we consider the standard reachability problem s.t. $f = \mathbf{F}g$ with g boolean and we apply straightforwardly (3) and the encodings of Table 1, we have:

$$[[\mathbf{F}g]]_k = \left((\neg \bigvee_{l=0}^{k} {}_lL_k \wedge \bigvee_{j=0}^{k} g_j) \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge \bigvee_{j=0}^{k} g_j) \right) . \tag{4}$$

The DAG structure allows for sharing the $\bigvee_{j=0}^{k} g_j$ and ${}_lL_k$ terms, but it cannot simplify (4) any further. On the other hand, we will see later that in this case $[[\mathbf{F}g]]_k = \bigvee_{j=0}^{k} g_j$, that is, all the ${}_lL_k$ terms can be dropped.

In general, there are often simplifications which can be done, e.g., by simply applying DeMorgan's rules and/or the associativity of conjuncts and disjuncts, or by recognizing properties due to the semantics of subformulas. In this paper we identified some of these simplifications, thus allowing for their application in the encoding algorithm. The identified simplifications aim to speed up the SAT solver in answering the submitted problem.

4 Optimizations to the encoding

Analyzing (1), (2), (3) and the inductive definitions of $[[f]]_k^i$ and ${}_l[[f]]_k^i$ in Table 1, we notice some properties which allow for introducing significant improvements in the size of the encodings. In the following we denote by $f \models g$ propositional model entailment.

4.1 Removing the "$\neg L_k$" component

Property 1. For all LTL formulas f and for all i, l, k s.t. $0 \leq i \leq k$ and $0 \leq l \leq k$,

$$[[f]]_k^i \models {}_l[[f]]_k^i \tag{5}$$

[2] Noteworthy cases of efficient implementation of DAGs are Reduced Boolean Circuits (RBCs) [1] and Boolean Expression Diagrams (BEDs) [16].

Proof. The result comes from the inductive definitions of $[[f]]_k^i$ and ${}_l[[f]]_k^i$ in Table 1, by induction on the structure of f. We recall that, in propositional logic, if $f_1 \models g_1$ and $f_2 \models g_2$, then $f_1 \wedge f_2 \models g_1 \wedge g_2$ and $f_1 \vee f_2 \models g_1 \vee g_2$, and $h \models h[g_1/f_1]^+$, where $h[g_1/f_1]^+$ is obtained by substituting positive occurrences of f_1 with g_1 in h.

- If $f \in \{p, \neg p\}$, then $[[f]]_k^i = {}_l[[f]]_k^i$.

We assume by inductive hypothesis that $[[h]]_k^i \models {}_l[[h]]_k^i$ and $[[g]]_k^i \models {}_l[[g]]_k^i$. Thus:

- $[[h]]_k^i \wedge [[g]]_k^i \models {}_l[[h]]_k^i \wedge {}_l[[g]]_k^i$ and $[[h]]_k^i \vee [[g]]_k^i \models {}_l[[h]]_k^i \vee {}_l[[g]]_k^i$.
- $[[\mathbf{X}g]]_k^i \models {}_l[[\mathbf{X}g]]_k^i$ and $[[\mathbf{G}g]]_k^i \models {}_l[[\mathbf{G}g]]_k^i$, as $\perp \models f$ for every f.
- $[[\mathbf{F}g]]_k^i \models {}_l[[\mathbf{F}g]]_k^i$, $[[h\mathbf{U}g]]_k^i \models {}_l[[h\mathbf{U}g]]_k^i$, $[[h\mathbf{R}g]]_k^i \models {}_l[[h\mathbf{R}g]]_k^i$, as they are all in the form " $[[f]]_k^i \models [[f]]_k^i[\mathbf{h}/{}_l\mathbf{h}, \mathbf{g}/{}_l\mathbf{g}]^+ \vee F^*$", where $[[f]]_k^i[\mathbf{h}/{}_l\mathbf{h}, \mathbf{g}/{}_l\mathbf{g}]^+$ is the formula obtained by substituting the positive occurrences of $[[h]]_k^j$'s and $[[g]]_k^j$'s with the respective ${}_l[[h]]_k^j$'s and ${}_l[[g]]_k^j$'s in $[[f]]_k^i$.

□

Property 2. The boolean expression $[[f]]_k$ defined in (3) is logically equivalent to

$$[[f]]_k^0 \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[[f]]_k^0). \tag{6}$$

Proof. If μ is a model for $(\neg L_k \wedge [[f]]_k^0) \vee \bigvee_{l=0}^{k}({}_lL_k \wedge {}_l[[f]]_k^0)$, then it is trivially a model also for $([[f]]_k^0 \vee \bigvee_{l=0}^{k}({}_lL_k \wedge {}_l[[f]]_k^0))$. Vice-versa, if μ is a model for $([[f]]_k^0 \vee \bigvee_{l=0}^{k}({}_lL_k \wedge {}_l[[f]]_k^0))$, then either it is a model for $\bigvee_{l=0}^{k}({}_lL_k \wedge {}_l[[f]]_k^0)$ or it is not. In the first case, μ is also a model for (3). In the second case, $\mu \models [[f]]_k^0$ and, for every l, $\mu \not\models ({}_lL_k \wedge {}_l[[f]]_k^0)$. From Property 1 for every l $\mu \models {}_l[[f]]_k^0$, thus $\mu \not\models {}_lL_k$. Therefore, $\mu \models \neg L_k$, and thus it is a model for (3).

□

Thus, in (3) the "$\neg L_k$" component is redundant and can be dropped.

4.2 Encodings ad hoc when $depth(f) \leq 1$

In this section we borrow from [13] some ideas from their encodings of CTL specifications of the form $\{\mathbf{AX}g, \mathbf{AG}g, \mathbf{AF}g, \mathbf{A}[h\mathbf{U}g], \mathbf{A}[h\mathbf{R}g]\}$, with h and g boolean, which we generalize to every LTL formula f s.t. $depth(f) \leq 1$. [3]

Property 3. If ${}_l[[f]]_k^0$ does not vary with l (we denote it by ${}_*[[f]]_k^0$), then we have:

$$[[f]]_k = [[f]]_k^0 \vee (L_k \wedge {}_*[[f]]_k^0). \tag{7}$$

[3] Notice that our encodings are dual w.r.t. those defined in [13] as here f is the negation of the specification.

Proof. From (6), we have:

$$
\begin{aligned}
[[f]]_k &= [[f]]_k^0 \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_*[[f]]_k^0) \\
&= [[f]]_k^0 \vee (\bigvee_{l=0}^{k} {}_lL_k \wedge {}_*[[f]]_k^0) \\
&= [[f]]_k^0 \vee (L_k \wedge {}_*[[f]]_k^0).
\end{aligned}
$$

□

Property 4. If ${}_l[[f]]_k^0$ does not vary with l, and if there exists a formula F_k^* such that ${}_l[[f]]_k^0$ can be rewritten as ${}_l[[f]]_k^0 = ([[f]]_k^0 \vee F_k^*)$, then we have:

$$
[[f]]_k = [[f]]_k^0 \vee (L_k \wedge F_k^*). \tag{8}
$$

Proof. Starting from (7), we can factorize the common term $[[f]]_k^0$:

$$
\begin{aligned}
[[f]]_k &= [[f]]_k^0 \vee (L_k \wedge ([[f]]_k^0 \vee F_k^*)) \\
&= [[f]]_k^0 \vee (L_k \wedge [[f]]_k^0) \vee (L_k \wedge F_k^*) \\
&= [[f]]_k^0 \vee (L_k \wedge F_k^*).
\end{aligned}
$$

□

As a particular case, if ${}_l[[f]]_k^0 = [[f]]_k^0$, then $F_k^* = \perp$ and thus $[[f]]_k = [[f]]_k^0$.

Property 5. If $depth(f) \leq 1$, then ${}_l[[f]]_k^0 = {}_*[[f]]_k^0$ does not vary with l. Thus, by Property 3 we have:

$$
[[f]]_k = [[f]]_k^0 \vee (L_k \wedge {}_*[[f]]_k^0). \tag{9}
$$

Proof. The result comes from Table 1 by induction on the structure of f.

- If $f \in \{p, \neg p\}$, then ${}_l[[f]]_k^i$ does not vary with l, for every i.
- If ${}_l[[h]]_k^i$ and ${}_l[[g]]_k^i$ do not vary with l, then neither do ${}_l[[h]]_k^i \wedge {}_l[[g]]_k^i$ and ${}_l[[h]]_k^i \vee {}_l[[g]]_k^i$, for every i.

If h and g are boolean, then ${}_l[[g]]_k^i$ and ${}_l[[h]]_k^i$ do not vary with l, for every i. Thus:

- If $i = 0$, then $i < k$ except when $k = 0$, in which case $l = i = k = 0$. Thus, ${}_l[[\mathbf{X}g]]_k^0$ does not vary with l.
- $min(0, l) = 0$, thus ${}_l[[\mathbf{G}g]]_k^0$ ${}_l[[\mathbf{F}g]]_k^0$ do not vary with l.
- as $i = 0$, the terms $\bigvee_{j=l}^{i-1} \ldots$ are null. Thus, ${}_l[[h\mathbf{U}g]]_k^0$ and ${}_l[[h\mathbf{R}g]]_k^0$ do not vary with l.

□

Thus, when $depth(f) \leq 1$, $[[f]]_k$ can be rewritten into the much simpler expression (9) or even into (8) if ${}_l[[f]]_k^0 = [[f]]_k^0 \vee F_k^*$ for some F_k^*.

f	$[[f]]_k^0$	$_*[[f]]_k^0$	$[[f]]_k$
g	g_0	g_0	g_0
$\mathbf{X}g$	g_1 $if\ k>0$ $\perp$ $otherwise.$	g_1 $if\ k>0$ g_0 $otherwise.$	g_1 $if\ k>0$ $_0L_0 \wedge g_0$ $otherwise.$
$\mathbf{G}g$	$\perp$	$\bigwedge_{j=0}^{k} g_j$	$L_k \wedge \bigwedge_{j=0}^{k} g_j$
$\mathbf{F}g$	$\bigvee_{j=0}^{k} g_j$	$\bigvee_{j=0}^{k} g_j$	$\bigvee_{j=0}^{k} g_j$
$h\mathbf{U}g$	$\bigvee_{j=0}^{k} \left(g_j \wedge \bigwedge_{n=0}^{j-1} h_n\right)$	$\bigvee_{j=0}^{k} \left(g_j \wedge \bigwedge_{n=0}^{j-1} h_n\right)$	$\bigvee_{j=0}^{k} \left(g_j \wedge \bigwedge_{n=0}^{j-1} h_n\right)$
$h\mathbf{R}g$	$\bigvee_{j=0}^{k} \left(h_j \wedge \bigwedge_{n=0}^{j} g_n\right)$	$\bigwedge_{j=0}^{k} g_j \ \vee$ $\bigvee_{j=0}^{k} \left(h_j \wedge \bigwedge_{n=0}^{j} g_n\right)$	$\bigvee_{j=0}^{k} \left(h_j \wedge \bigwedge_{n=0}^{j} g_n\right) \vee$ $(L_k \wedge \bigwedge_{j=0}^{k} g_j)$

Table 2. $[[f]]_k^i$, $_*[[f]]_k^i$ and $[[f]]_k$, $f \in \{g, \mathbf{X}g, \mathbf{G}g, \mathbf{F}g, h\mathbf{U}g, h\mathbf{R}g\}$, h, g boolean.

Example 1. Consider the LTL model checking problem $M \models \mathbf{A}((h\mathbf{U}g) \rightarrow \mathbf{G}p)$, and its corresponding BMC problem $M \models_k \mathbf{E}f$, f being $((h\mathbf{U}g) \wedge \mathbf{F}\neg p)$, h, g and p being boolean. We have $[[f]]_k^0 = {}_l[[f]]_k^0 = \bigvee_{j=0}^{k} \left(g_j \wedge \bigwedge_{n=0}^{j-1} h_n\right) \wedge \bigvee_{j=0}^{k} \neg p_j$, $F^* = \perp$, thus from (8), $[[f]]_k = [[f]]_k^0$.

$\diamond$

This is not a formula of the kind addressed by [13]. However, if we restrict to $f \in \{g, \mathbf{X}g, \mathbf{G}g, \mathbf{F}g, h\mathbf{U}g, h\mathbf{R}g\}$ with h, g boolean, then we can apply (9) and obtain the same results as in [13], as shown in Table 2.

As final remark we can notice that, when $depth(f) > 1$ it is in general the case that $[[f]]_k^i$ is not a subformula of ${}_l[[f]]_k^i$. This property can be exploited to simplify the work of the SAT solver as it will be shown in Section 5.

4.3 Handling Fairness constraints: $\mathbf{f = GFg}$, g boolean

We consider here the encoding of fairness constraints, that is, LTL formulas in the form "$\mathbf{GF}g$", g being a boolean formula. From Table 1 we have that $[[\mathbf{GF}g]]_k^i = \perp$ and that

$$ {}_l[[\mathbf{GF}g]]_k^0 = \bigwedge_{i=0}^{k} \bigvee_{j=min(i,l)}^{k} g_j. \tag{10}$$

We subdivide the external conjunction in two parts: for $i < l$, —outside the loop— and for $i \geq l$ —inside the loop. Inside the first conjunct, we further subdivide the disjunction in two parts: for $i < l$, —outside the loop— and for $i \geq l$ —inside the loop (11). The underlined term $\bigvee_{j=l}^{k} g_j$ in both conjuncts of (11) does not vary with i, thus we can take it out from their respective conjunctions:

$$ {}_l[[\mathbf{GF}g]]_k^0 = \bigwedge_{i=0}^{l-1} \left(\bigvee_{j=i}^{l-1} g_j \vee \underline{\bigvee_{j=l}^{k} g_j} \right) \quad \wedge \quad \bigwedge_{i=l}^{k} \underline{\bigvee_{j=l}^{k} g_j} \tag{11}$$

$$= \left(\bigwedge_{i=0}^{l-1} \bigvee_{j=i}^{l-1} g_j \vee \underline{\bigvee_{j=l}^{k} g_j} \right) \quad \wedge \quad \underline{\bigvee_{j=l}^{k} g_j}$$

$$= \bigvee_{j=l}^{k} g_j. \tag{12}$$

Intuitively, ${}_l[[\mathbf{GF}g]]_k^0$ holds if and only if g_j holds in at least one of the internal states of the loop. From (3) and (12) we have thus:

$$[[\bigwedge_r \mathbf{GF}g^{(r)} \wedge f]]_k = \bigvee_{l=0}^{k} \left({}_lL_k \wedge \bigwedge_r \bigvee_{j=l}^{k} g_j{}^{(r)} \wedge {}_l[[f]]_k^0 \right) \tag{13}$$

which represents the case of bounded model checking $M \models_k \mathbf{E}f$ under the set of fairness constraints $\{\mathbf{GF}g^{(r)}\}_r$. Intuitively, (13) means "there is a loop in which f holds s.t., for each $g^{(r)}$, there is a state s_j in the loop in which $g^{(r)}$ holds". Again, if ${}_l[[f]]_k^0$ does not depend on l, it can be extracted from the disjunction:

$$[[\bigwedge_r \mathbf{GF}g^{(r)} \wedge f]]_k = {}_*[[f]]_k^0 \wedge \bigvee_{l=0}^{k} \left({}_lL_k \wedge \bigwedge_r \bigvee_{j=l}^{k} g_j{}^{(r)} \right). \tag{14}$$

Notice that, if we have only one fairness constraint $\mathbf{GF}g$, we can rewrite (13) as:

$$\begin{aligned}
[[\mathbf{GF}g \wedge f]]_k &= \bigvee_{l=0}^{k} \left(\bigvee_{j=l}^{k} g_j \wedge {}_lL_k \wedge {}_l[[f]]_k^0 \right) \\
&= \bigvee_{l=0}^{k} \bigvee_{j=l}^{k} (g_j \wedge ({}_lL_k \wedge {}_l[[f]]_k^0)) \\
&= \bigvee_{j=0}^{k} \bigvee_{l=0}^{j} (g_j \wedge {}_lL_k \wedge {}_l[[f]]_k^0) \\
&= \bigvee_{j=0}^{k} \left(g_j \wedge \bigvee_{l=0}^{j} ({}_lL_k \wedge {}_l[[f]]_k^0) \right).
\end{aligned} \tag{15}$$

Intuitively, (15) means "there is a state s_j in which g holds, s.t. there is a loop containing s_j in which f holds". This means lifting to the top of the formula the boolean constraint g_j –which typically come straightforwardly from primary inputs.

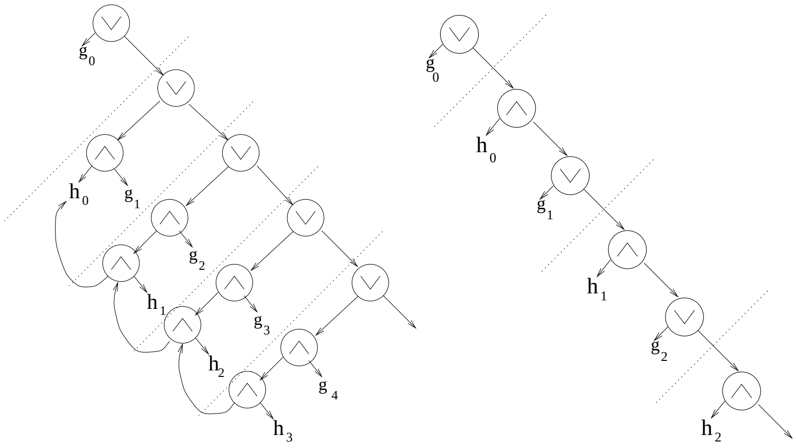

Fig. 1. $[[h\mathbf{U}g]]_k^0 = {}_l[[h\mathbf{U}g]]_k^0$, g, h boolean. Left: as in (16). Right: as in (17).

4.4 "Tableau-style" encodings for $\mathbf{f} = \mathbf{hUg}$ and $\mathbf{f} = \mathbf{hRg}$

Consider $f = h\mathbf{U}g$, h and g being generic LTL formulas. From Table 1, for $i = 0$ we have that

$$[[h\mathbf{U}g]]_k^0 = \begin{array}{lllllr} (& [[g]]_k^0 & & & & \lor \\ ((& [[g]]_k^1 \land & & [[h]]_k^0 & &) \lor \\ ((& [[g]]_k^2 \land & (& [[h]]_k^0 \land & [[h]]_k^1 \;) &) \lor \\ ((& [[g]]_k^3 \land & ((& [[h]]_k^0 \land & [[h]]_k^1 \;) \land \; [[h]]_k^2 \;) &) \lor \\ & \dots & & & & \dots \\ ))))\dots)) & & & & & \end{array} \tag{16}$$

we notice that each $[[h]]_k^i$ is a common conjunct from the $i + 2$-th conjunction onward. Thus, factorizing iteratively the $[[h]]_k^i$'s, we obtain:

$$[[h\mathbf{U}g]]_k^0 = \begin{array}{llll} (\; [[g]]_k^0 \lor & & & \\ \; (\; [[h]]_k^0 \land (\; [[g]]_k^1 \lor & & & \\ & (\; [[h]]_k^1 \land (\; [[g]]_k^2 \lor & & \\ & & (\; [[h]]_k^2 \land (\; [[g]]_k^3 \lor & \\ \quad \dots & \dots & \dots & \\ ) \,)) &)) &)) & \dots \end{array} \tag{17}$$

Analogous transformations can be done for $[[h\mathbf{U}g]]_k^i$, ${}_l[[h\mathbf{U}g]]_k^0$ and ${}_l[[h\mathbf{U}g]]_k^i$.

A comparison of the (DAG of the) encodings in (16) and (17), with h and g boolean, is represented in Figure 1. The second encoding requires about $2 \cdot k$ new nodes, while the first, even with the best factorization, requires about $3 \cdot k$ nodes.

Intuitively, the encoding (17) can be seen as a straightforward application of the recursive expansion:

$$h\mathbf{U}g = g \lor (h \land \mathbf{X}(h\mathbf{U}g)), \tag{18}$$

which is the basis of the tableau encoding of LTL formulas into automata [15]. For this reason, we call this kind of encodings "tableau-style". Analogous encodings can be produced for $[[h\mathbf{R}g]]_k^i$ and ${}_l[[h\mathbf{R}g]]_k^i$.

The tableau-style encodings are logically equivalent to those of Table 1, thus properties 1 and 2 still hold and, if $depth(f) \leq 1$, then we still have that ${}_*[[h\mathbf{U}g]]_k^0 = [[h\mathbf{U}g]]_k^0$ and ${}_*[[h\mathbf{R}g]]_k^0 = \bigwedge_{j=0}^k g_j \vee [[h\mathbf{R}g]]_k^0$. As a consequence, the optimizations described in Sections 4.1 and 4.2 apply to tableau-style encodings as well.

5 Adding implicit constraints

Property 1 suggests a further optimization to apply to (3) to speed up the work of the SAT solver. In Section 4.2 we noticed that if $depth(f) > 1$, then $[[f]]_k^i$ is not necessarily a subformula of ${}_l[[f]]_k^i$. Thus, when the SAT solver has assigned (the labeling variable of) $[[f]]_k^i$ to true, in general it may need extra search to infer that the ${}_l[[f]]_k^i$'s are true; vice-versa, when it has assigned one ${}_l[[f]]_k^i$ to false, it may need extra search to infer that $[[f]]_k^i$ is false.

Thus, the idea is to add a series of constraints to the resulting DAG of $[[M, f]]_k$, to speed up the search. If g is a subformula of f s.t. $depth(f) > depth(g) > 1$, then for every i and l such that $[[g]]_k^i$ and ${}_l[[g]]_k^i$ occur in the DAG of $[[M, f]]_k$, the subformula:

$$\neg\ [[g]]_k^i \vee {}_l[[g]]_k^i \tag{19}$$

is added to the DAG of $[[M, f]]_k$.

As both $[[g]]_k^i$ and ${}_l[[g]]_k^i$ already occur in the DAG, the subformula (19) is simply a binary clause in the labeling variables of $[[g]]_k^i$ and ${}_l[[g]]_k^i$. Thus, when the SAT solver has assigned (the labeling variable of) $[[g]]_k^i$ to true, then it assigns to true also all the ${}_l[[g]]_k^i$'s by simple unit propagation, and vice-versa. If $depth(f) = depth(g) > 1$, then only the ${}_l[[g]]_k^0$'s occur in the DAG of $[[M, f]]_k$, thus only the constraints in $\neg\ [[g]]_k^0 \vee {}_l[[g]]_k^0$ are added, for all l. On the whole, this corresponds to add to the DAG of $[[M, f]]_k$ the formula

$$\bigwedge_{\substack{g \subseteq f:\\ depth(g) =\\ depth(f) > 1}} \bigwedge_{l=0}^{k} (\neg\ [[g]]_k^0 \vee {}_l[[g]]_k^0) \wedge \bigwedge_{\substack{g \subset f:\\ depth(f) >\\ depth(g) > 1}} \bigwedge_{i=0}^{k} \bigwedge_{l=0}^{k} (\neg\ [[g]]_k^i \vee {}_l[[g]]_k^i)\,, \tag{20}$$

which corresponds to add $O(k^2 \cdot |f|)$ binary constraints.

6 Exploiting the associativity order

The main reason why we use DAGs to represent propositional formulas is that they allow for sharing a lot of subformulas, reducing thus the size and number of extra variables of the resulting CNF-ized formula submitted to the SAT solver [12, 10]. Unfortunately, using DAG representation does not help to recognize as identical two formulas which differ only modulo associativity of $\wedge, \vee$, like, e.g., $(p \wedge (q \wedge r))$ and $(p \wedge q) \wedge r))$. When encoding complex LTL formulas the problem becomes very relevant, and it requires some care.

Consider for example the case of $f = h\mathbf{U}g$, with h and g boolean, and consider the j-th disjunct $\bigwedge_{n=0}^{j-1} h_n$ in Table 2. If the conjuncts are associated left-to-right:

$$j: \quad (h_1 \wedge (h_2 \wedge (\ldots \wedge (h_{j-2} \wedge h_{j-1}))) \ldots))) \tag{21}$$

$$j+1: \quad (h_1 \wedge (h_2 \wedge (\ldots \wedge (h_{j-2} \wedge (h_{j-1} \wedge h_j)))) \ldots))), \tag{22}$$

then the DAGs cannot share any sub-formula of the conjunction. If, instead, the conjuncts are associated right-to-left:

$$j: \quad ((((\ldots (h_1 \wedge h_2) \ldots \wedge h_{j-2}) \wedge h_{j-1}) \tag{23}$$

$$j+1: \quad ((((\ldots (h_1 \wedge h_2) \ldots \wedge h_{j-2}) \wedge h_{j-1}) \wedge h_j) \tag{24}$$

then the DAGs share the components $((((\ldots (h_1 \wedge h_2) \ldots \wedge h_i)$, as in Figure 1 (left). If we consider instead the example of $f = \mathbf{GF}g$ with g boolean (13), in order to let the DAG share the common disjuncts, the terms $G_l^k = \bigvee_{j=l}^{k} g_j$ must be associated in the opposite way:

$$G_l^k: \quad (g_l \vee (g_{l+1} \vee \ldots (g_{k-2} \vee (g_{k-1} \vee g_k)) \ldots)))) \tag{25}$$

$$G_{l+1}^k: \quad (g_{l+1} \vee \ldots (g_{k-2} \vee (g_{k-1} \vee g_k)) \ldots))). \tag{26}$$

Thus, using DAGs with more complex LTL formulas, it is very important to decide each time the best associativity order of the conjuncts to maximize the sharing of common nodes by DAGs.

7 Conclusions and Future Works

In this paper we identified some simplifications of the encoding of bounded model checking problems into propositional satisfiability problems. These simplifications aim to reduce the effort of the SAT solvers in this problem. We are currently integrating the defined optimizations within NuSMV. Preliminary experiments on the problems proposed in [2] (not reported here for lack of space) confirm that these optimizations lead to a reduction on the size of the CNF formulas submitted to the SAT solver, and to a significant reduction in the time required by the SAT solver to return an answer.

Future work goes in two main directions. The first one consists in completing the integration of all the simplifications defined in this paper within NuSMV. The second direction consists in performing an exhaustive experimental analysis aimed, from one hand, to show the effectiveness of the devised simplifications, and from the other hand, to possibly discover new ones. A crucial point to perform a detailed experimental analysis is the lack of a standard benchmark suite for evaluating the performances of the encoding algorithms. As part of this task, we are working on the definition of a benchmark suite for bounded model checking problems.

References

1. P. A. Abdullah, P. Bjesse, and N. Een. Symbolic Reachability Analysis based on SAT-Solvers. In *Sixth Int.nl Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, 2000.
2. A. Biere, A. Cimatti, E. M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *Proc. TACAS'99*, pages 193–207, 1999.
3. A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifying safety proeprties of a power pc microprocessor using symbolic model checking without BDDs. In *Proc CAV99*, volume 1633 of *LNCS*, pages 60–71, Berlin, 1999. Springer.
4. R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.
5. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, June 1992.
6. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV : a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4), March 2000.
7. E. Clarke, O. Grumberg, and D. Long. Model Checking. In *Proceedings of the International Summer School on Deductive Program Design*, Marktoberdorf, Germany, 1994.
8. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
9. F. Copty, L. Fix, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of Bounded Model Checking at an Industrial Setting. In *Proc. CAV'2001*, LNCS, Berlin, 2001. Springer.
10. E. Giunchiglia and R. Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In *Proc. AI*IA'99*, number 1792 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1999.
11. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.
12. D.A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.
13. D. Sheridan and T. Walsh. Clause Forms Generated by Bounded Model Checking. In *Proc. Eighth Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, University of York, March 2001.
14. O. Shtrichmann. Tuning SAT checkers for bounded model checking. In *Conference of Computer Aided Verification*, volume 1855 of *LNCS*, pages 480–494, Berlin, 2000. Springer.
15. M. Y. Vardi and P. Wolper. Automata-Theoretic Techniques for Modal Logics of Programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.
16. P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In *Proc. CAV'2000*, volume 1855 of *LNCS*, pages 124–138, Berlin, 2000. Springer.